Common Fortify findings in jQuery

This page has been made public for vendors

Question

When scanning my code, I am seeing findings like Insecure Randomness and Dynamic Code Injection identified in jQuery. How can I audit these findings?

Answer

It is important to audit all findings identified by Fortify to determine if they pose a risk to the application, including those in third party libraries like jQuery. See this technical note on auditing third party code for more general information on this topic.

The following recommendations apply to jQuery versions 1.x - 3.1.1

Insecure Randomness

These are usually false positives in jQuery.

The main concern with Insecure Randomness is when a pseudo-random number generator (PRNG) is used for cryptography or in a security context. If the PRNG is not used for cryptography, then it is likely not an issue. For example, in jQuery, random numbers are used for several reasons such as generating a unique version number and for unique element identifiers. See the following example where jQuery uses the insecure `Math.random()` to generate a unique version number:

jQuery use of Math.random()

```
jQuery.extend( {
 // Unique for each copy of jQuery on the page
expando:
 "jQuery" + ( version + Math.random() ).replace(
/\D/g, "" ),
 ...
} );
```

In this case, the PRNG is not used in a security context so it is not a concern. This kind of analysis ensures the application is safe regardless of whether or not this portion of code is in use by the application.

Dynamic Code Evaluation

These are often false positives in jQuery.

The concern with Dynamic Code Evaluation findings is when unvalidated user input (or any data external to the application) is interpreted as code by the application. In the following example, jQuery calls `setTimeout()` which executes a function after a specified amount of time. Since `setTimeout()` evaluates the code provided to it, it could potentially be used to execute malicious code. In this case jQuery defines an anonymous function that aborts the jQuery XMLHttpRequest (`jqXHR`) after a timeout:

HPE Fortify Version	16.20 and later
Programming Language	☐ C/C++ ☐ .NET ☐ Java ☐ Objective-C ✓ Other
Fortify Audit Workbench	✓ Yes ☐ No
Fortify IDE Plugin	✓ Yes ☐ No
Other Fortify Component	☐ Yes ☐ No

Request code review tools, validations, and support HERE.

jQuery use of setTimeout()

```
// Timeout
if ( s.async && s.timeout > 0 ) {
 timeoutTimer = window.setTimeout( function() {
  jqXHR.abort( "timeout" );
 }, s.timeout );
}
```

Since there is no user or external input being used in the call to `setTimeout()`, this finding is not a concern. **Note that this only applies to Dynamic Code Evaluation findings that involve calls to `setTimeout()` similar to the above example. Other findings need to be evaluated on a case-by-case basis.**

JavaScript Hijacking

These need to be reviewed in the context of how the application is using jQuery.

Applications that use JavaScript Object Notation (JSON) to transport sensitive data can be vulnerable to JavaScript Hijacking, so these findings need to be evaluated on a case-by-case basis. If JSON is being used to transport confidential information, it's possible that a loophole in the Same Origin Policy could allow JavaScript from one site to be executed in the context of another site. An attacker would then be able to take advantage of this by witnessing the results of their code being run on the vulnerable site. See the Details and Recommendations tabs in Fortify as well as the resources below for more information.

References

- VA Software Assurance Wiki - Auditing Third Party Code
- OWASP - Insecure Randomness
- OWASP - Dynamic Code Evaluation
- CAPEC/MITRE - JSON Hijacking
- jQuery